# EXHIBIT E

# 5.0 Reference

This section describes the actual OLE Interfaces exposed, the definitions of the data structures used when passing data around, and the definitions of each class used internally by the stream.

## 5.1 Interfaces

Other than the standard IUnknown and IClassFactory interfaces, the stream component exposes two custom interfaces: the IXMC_Stream and IXMC_StreamInit interfaces. The diagram below graphically displays what the stream objects looks like.

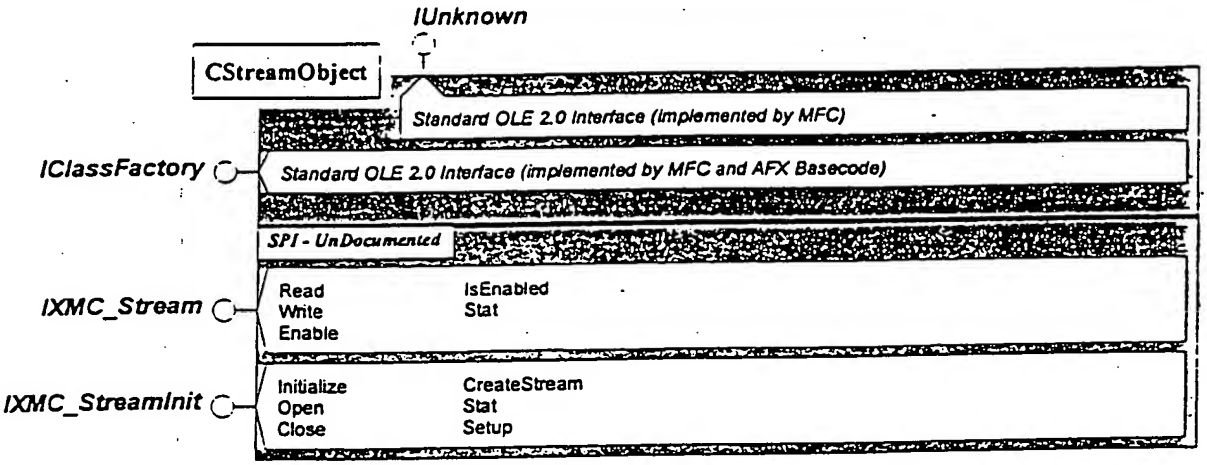


Figure 12 *Interface-Map.*

Each of the following sections describes the custom interfaces exposed by the stream.

### 5.1.1 IXMC_StreamInit Interface

The following methods are exposed by the IXMC_StreamInit interface.

```
IXMC_StreamInit Interface
{
        HRESULT Initialize( HSTREAM hStream, LPSTREAM_INFO lpSI );
        HRESULT Open( void );
        HRESULT Close( void );
        HRESULT CreateStream( LPXMCSTREAM lpStream );
        HRESULT Stat( LPSTREAM_INFO lpSI );
        HRESULT Setup( LPSTREAM_INFO lpSI );
}
```

### 5.1.2 IXMC_Stream Interface

The following methods are exposed by the IXMC_Stream interface.

```
IXMC_Stream Interface
{
        HRESULT Write( LPTSTR pszBuffer, DWORD dwSize );
        HRESULT Read( LPTSTR pszBuffer, DWORD dwMax, LPDWORD lpdwRead );
        HRESULT Enable( BOOL bEnable );
        HRESULT IsEnabled( void );
        HRESULT Stat( LPSTREAM_INFO lpSI );
}
```

## 5.2 Exported Functions

The following are the functions exported by the stream DLL.

```
XMC_STREAM_MODULETYPE       DLLGetModuleType( void );
LPCLSID                     DLLGetCLSID( void );
BOOL                        DLLRegisterServer( void );
BOOL                        DLLUnRegisterServer( void );
```

## 5.3 Structures and Defines

This section defines all structures, enumerations, and defines used by the stream.

### 5.3.1 XMC_STREAM_MODULETYPE Enumeration

This enumeration defines the type of streams available. Each stream must return its type when the user calls the exported **DLLGetModuleType** function.

```
enum XMC_STREAM_MODULETYPE
{
        XMC_STREAM_MT               = 0x3000,
        XMC_STREAM_MT_PCBUS         = 0x3001,
        XMC_STREAM_MT_SERIAL        = 0x3002,
        XMC_STREAM_MT_TEXTFILE      = 0x3003,
        XMC_STREAM_MT_DBGMON        = 0x3004,
        XMC_STREAM_MT_CUSTOM        = 0x3005
};
```

### 5.3.2 XMC_STREAM_INFO Structure

This structure is used to pass all stream specific data used to both setup the stream, and query the stream for its current settings.

```
struct XMC_STREAM_INFO
{
        XMC_HSTREAM                 m_hStream;
        XMC_STREAM_MODULETYPE       m_mt;

        union {
                struct PCBus
                {
                        DWORD dwPort;
                        DWORD dwIRQ;
                },

                struct Serial
                {
                        DWORD dwPort;
                        DWORD dwBPS;
                },

                struct TextFile
                {
                        LPCTSTR pszFileName;
                },

                struct Custom
                {
                        DWORD dwParam1;
                        DWORD dwParam2;
                },
        };
};
```

## 5.4 Classes

As seen in Chapter 3.0 Object Interaction-Map, there are four main C++ classes used to implement the stream component: the **CStreamInitDisp**, **CStreamDisp**, **CRegistryMgr**, **CIOMgr**, and **CIOHAL** classes. The definition of each class is described below.

### 5.4.1 CStreamInitDisp Class

The purpose of the CStreamInitDisp class is to act as a dispatcher that funnels calls from the stream component out into the other C++ objects in the system. If viewed with a control-model-view paradigm, the CStreamInitDisp would be considered a control object. The following is the definition of the CStreamInitDisp object.

```
class CStreamDisp
{
      public:
            //---- Constructors & Destructors ----

            CStreamInitDisp( void );
            ~CStreamInitDisp( void );

            //---- IXMC_StreamInit Actions ----

            DWORD Initialize( HSTREAM hStream, LPSTREAM_INFO lpSI );
            DWORD Open( void );
            DWORD Close( void );
            DWORD Stat( LPXMCSTREAM_INFO lpSI );
            DWORD Setup( LPXMCSTREAM_INFO lpSI );


            //---- IXMC_Stream Actions ----

            DWORD Write( LPTSTR pszBuffer, DWORD dwSize );
            DWORD Read( LPTSTR pszBuffer, DWORD dwMax, LPDWORD lpdwRead );
            DWORD Enable( BOOL bEnable, LPBOOL lpbEnable );
            BOOL  IsEnabled( void );

      private:
            //---- Private Data ----

            BOOL              m_bEnabled;
            XMCSTREAMHANDLE   m_hStream;
            XMC_STREAM_INFO   m_StreamInfo;
};
```

### 5.4.2 CRegistryMgr Class

The CRegistryMgr manages all interactions taking place with the registration database. Included in such actions, are storing and loading all state data at the stream handle location. Below, is the definition of the CRegistryMgr class.

```
class CRegistryMgr
{
      public:
            //---- Constructors & Destructors ----

            CRegistryMgr( void );
            ~CRegistryMgr( void );

            //---- Actions ----

            DWORD Initialize( HSTREAM hStream );
            DWORD StoreSettings( LPXMCSTREAM_INFO lpSI );
            DWORD LoadSettings( LPXMCSTREAM_INFO lpSI );

      private:
            //---- Private Data ----

            XMCSTREAMHANDLE      m_hStream;
};
```

### 5.4.3 CIOMgr Class

The CIOMgr manages the CIOHAL object used to directly control the target during reading and writing operations. Most other operations pass directly through to the CIOHAL object. For example, if the underlying hardware does not support large blocks of data, the CIOMgr takes care of buffering the data and sending smaller, more manageable pieces, to the CIOHAL. The following is the definition of the CIOMgr class.

```
class CIOMgr
{
      public:
            //---- Constructors & Destructors ----

            CIOMgr( void );
            ~CIOMgr( void );

            //---- Initialization ----

            DWORD Initialize( LPXMCSTREAM_INFO lpSI );
            DWORD Register( LPCTSTR pszDriverName );

            //---- Initialization ----

            DWORD Open( void );
            DWORD Close( void );
            DWORD Read( LPTSTR pszBuf, DWORD dwMax, LPDWORD lpdwRead );
            DWORD Write( LPTSTR pszBuf, DWORD dwSize );

      private:
            //---- Private Data ----

            XMCSTREAM_INFO      m_StreamInfo;
};
```

### 5.4.4 CIOHAL Class

The **CIOHAL** is the lowest level of the stream component. This object directly communicates with either the target itself, or a device driver used to communicate with the target. Since the code in this module is specific to the communication protocol used during all data transmissions with the target, it must be rewritten for each stream. The following, is a class definition for the **CIOHAL** object.

```
class CIOHAL
{
      public:
            //---- Constructors & Destructors ----

            CIOHAL( void );
            ~CIOHAL( void );

            //---- Initialization ----

            DWORD Initialize( LPXMCSTREAM_INFO lpSI );
            DWORD Register( LPCTSTR pszDriverName );

            //---- Initialization ----

            DWORD Open( void );
            DWORD Close( void );
            DWORD Read( LPTSTR pszBuf, DWORD dwMax, LPDWORD lpdwRead );
            DWORD Write( LPTSTR pszBuf, DWORD dwSize );

      private:
            //---- Private Data ----

            XMCSTREAM_INFO      m_StreamInfo;
};
```

You may notice that this class definition is almost identical to the **CIOMgr** class. The reason for the similarity is that the **CIOHAL** is the workhorse that the **CIOMgr** class drives. If you looked at the code, you would see that the only difference between the two is in the Read and Write methods. **CIOHAL** does not support any buffering of data, where **CIOMgr** does.

## *5.5 Registration Database Data*

Each stream stores its data persistantly in the registration database. Below, is the reg data format.

```
XMC.Stream.100
    |---- 0x00000001
    |           |---- Type          = "ST_PCBUS"
    |           |---- dwPort        = 0x0730
    |           |---- dwIRQ         = 0x0010
    |
    |---- 0x00000002
    |           |---- Type          = "ST_SERIAL"
    |           |---- dwPort        = "COM1"
    |           |---- dwBPS         = 9600
    |
    |---- 0x00000003
    |           |---- Type          = "ST_TEXTFILE"
    |           |---- pszFileName   = "c:\temp\foo.log"
    |
    |---- 0x00000004
    |           |---- Type          = "ST_DBGMON"
    |
    |---- 0x00000005
                |---- Type          = "ST_CUSTOM"
                |---- dwParam1      = 0
                |---- dwParam2      = 0
```